Data Structure & algorithm

- **Binary Problems**
- **2-Pointer Approach**
- **Palindrome**
- **Rotation of Array**
- **Dry run of solutions**

#Vale_freeConten
t

SWIPE

Follow
for more updates

Write the code using recursion

```python
def moves(n, s, d, a):

    #Base condition
    if n == 1:
        return 1

    #logic
    # moves n-1 s->a, then moves 1 biggest coin s->d, then moves n-1 coins a->d
    return moves(n-1,s,a,d) + 1 + moves(n-1,a,d,s)

#Driver code
print(moves(1,'s','d','a'))
print(moves(2,'s','d','a'))
print(moves(3,'s','d','a'))
print(moves(4,'s','d','a'))
```

```
1
3
7
15
```

Q Solve this linear search problem through recursion.

```python
def search(arr, num):
    for i in range(len(arr)):
        if arr[i] == num:
            return i
    return -1
```

Ans → def search (arr, sI, data):

arr, sI, data

↳ for ex → search([5,4,8,10,12,9], 2, 9)

When we pass arr → we are trying to search in this index [2 → till last]

to see 9 is present OR not

```python
#Ans:

def search_rec(arr, sI, data):
    #Base condition
    if sI >= len(arr):
        return "Value does not present in this array"

    if arr[sI] == data:
        return sI
    else:
        return search_rec(arr, sI+1, data)

# Driver code:
print('Value present at ',search_rec([2,3,4,34,22,54,9], 2, 22),'th index.')
```

```
Value present at  4 th index.
```

Q This is binary problem solve this through Recursion.

```python
def binary_search(arr, num):
    left = 0
    right = len(arr)-1

    #mid = (left + right)/2


    while (left <= right):
        mid = int(left + (right-left)/2)   ## this will take less space
        if arr[mid] == num:
            return mid
        elif arr[mid] > num:
            right = mid-1
        else:
            left = mid+1
    return -1
```

Soln → This is binary search problem → Already Sorted arry

for Recursion we need →

1. Mid { a. Start index, b. end index }
2. arr
3. data

```python
def binary_search(arr, left, right, num):
    # Base condition
    if left > right :
        return 'Value not present in this array'

    #Logic
    mid = int(left + (right-left)/2)

    if arr[mid] == num:
        return mid

    elif arr[mid] > num:
        return binary_search(arr, left, mid-1, num)
    else:
        return binary_search(arr, mid+1, right, num)

# Driver code
print(binary_search([2,4,6,8,10], 0, 5, 8))
print(binary_search([2,4,6,8,10], 0, 5, 3))  #checking for 3
```

```
3
Value not present in this array
```

Dry Run →

the no. that trying to find

binary([1,3,5,9,11], 0, 4, 9)

① arr = [1,3,5,9,11], left = 0, right = 4, num → 9
(0 1 2 3 4)
↳ mid = 2

② arr = [1,3,5,9,11], left → 3, right → 4, num → 9
(0 1 2 3 4)
↳ mid → 3 + 1/2 → 3.5 → (3)

3rd index

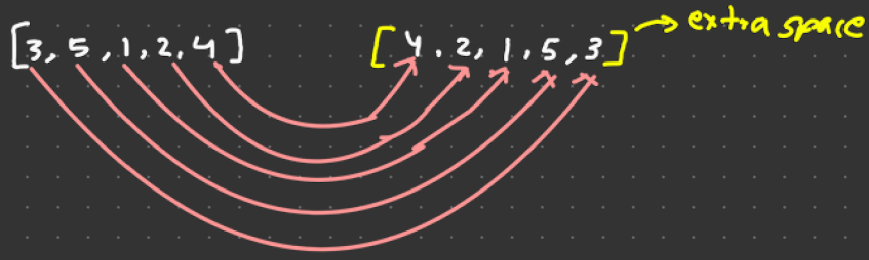
Ans → (3)

T(n) → $n, \frac{n}{2}, \frac{n}{4}, \ldots$

T(n) → O(log n)

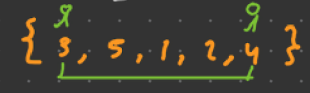
Google : You have an arr = [3,5,1,2,4], this is a immutable fun. Create another array, which is Reverse of it.

↳ Easy Method → Using Extra space :-



2 pointer approach →

[3,5,1,2,4]



[∵ Replace the pointer element]

Reverse by using 2 pointer approach

Code :-

```
def reverseArr(arr):
    i = 0
    j = len(arr)-1

    while(i<j):
        # swap elements at i and j index
        arr[i],arr[j] = arr[j],arr[i]
        i += 1
        j -= 1

#driver code
arr = [3,5,1,2,4]
reverseArr(arr)
print(arr)
```

```
[4, 2, 1, 5, 3]
```

Dry run →

reverseArr([3,5,1,2,4])

arr = [3,5,1,2,4]
0 1 2 3 4 → index

i = 0, j = 5-1 → 4

[4,5,1,2,3]
0 1 2 3 4 → index

i = 1, j → 3

[4,2,1,5,3]
0 1 2 3 4 → index

i = 2, j → 2

Here while loop terminate

Because i < j condition brake here

Final Result ⇒ [4,2,1,5,3]

→ Try to solve this using recursion

```python
# Trying to solve this problem using recursion

def reverse_rec(arr, sI, eI):
    # base condition
    if sI >= eI:
        return

    #Logic
    arr[sI], arr[eI] = arr[eI], arr[sI]  #swapping

    #Recursion
    reverse_rec(arr, sI+1, eI-1)
#driver code
arr = [3,5,1,2,4]
reverse_rec(arr, 0, len(arr)-1)
print(arr)
```

```
[4, 2, 1, 5, 3]
```

when we have immutable array which prefer ↴

Simple linear search ✓

Vs

Recursion

Q Is time complexity does any dependecy on recursion or loop?

↳ Time complexty always about

No. of operation ∝ Input size

When we are solving any problem for TC() → How my no. of operation are relay on input size.

Is is linear, logerathimic, Quradtic

that depends on the logic that we actually given that thing decide what type of complexity is it.

Palindrome

[1,2,3] → Same from right / Same from left

[1,2,3,3,2,1] → Palindrome ✓

[4] → single element also consider → Palindrome ✓

↳ Create a code that give True / False when arr = palindrome

Here we are using 2 Pointer approach

```python
# Ans:

def isPalindrome(arr):
    # 2 pointer approch
    left = 0
    right = len(arr)-1

    while(left < right):
        if arr[left] != arr[right]:
            return False
        left += 1
        right -= 1
    return True

#driver code
arr = [1,2,3,3,2,1]
arr1 = [1,2,4]
arr2 = [5]

print(isPalindrome(arr))
print(isPalindrome(arr1))
print(isPalindrome(arr2))
```

```
True
False
True
```

Dry Run :-

isPalindrome([1,2,2,1])

① arr = [1,2,2,1], left → 0, right → 3 (4-1), comparision

② arr = [1,2,2,1], left → 1, right → 2

③ arr = [1,2,2,1], left → 2, right → 1

Here while loop breake → left < right

④ → return → True

Rotation of array

- Rotation of array is that where all the position of element shifted left to right (∵ L → R). In this the last element comes to the first position.

Ex →

Rotate this array at once (L → R)

[3, 1, 2, 5, 4]

↳ [4, 3, 1, 2, 5] Rotate once again

↳ [5, 4, 3, 1, 2]

We gave you a array, try to rotate it K-times (L → R) and after rotation tell us How the final array look like.

↳ Understand the concept,

1st → k = 2, arr = [1, 2, 3, 4], then ?

↳ [4, 1, 2, 3]

↳ [3, 4, 1, 2]

our end goal is to find this array

Rotation Theory

arr = [1,2,3,4,5] → Rotate it 3 time.

↳ ① Step :- Reverse the whole array :- [5,4,3,2,1]

② Step :- Reverse k^{th} element arry :- [5,4,3,2,1]
(∵ k → 3 → 3 element)

⇒ [3,4,5,2,1]

③ Step :- Reverse (n−k) element :- [3,4,5,2,1]

[3,4,5,1,2]

Code :-

```
def rotate(arr,k):
    k = k%len(arr) -> 2%3 => 2

    for i in range(k):              ## Num of times the rotation has to take place

        #LOGIC
        prev = arr[0] -> 7
        curr = arr[0] -> 7
        arr[0] = arr[len(arr)-1]
        j = 1                  10
                               1,2,3   3
        while(j < len(arr)):
            curr = arr[j]-> 9 , 10
            arr[j] = prev-> 7 , 9
            prev = curr -> 9 , 10
            j += 1      -> 2 , 3

#drive code
arr1 = [1,2,3,4,5]
arr2 = [1,2,3,4,5]
arr3 = [1,2,3,4,5]

rotate(arr1,1)
print(arr1)
rotate(arr2,2)
print(arr2)
rotate(arr3,3)
print(arr3)
```

```
[5, 1, 2, 3, 4]
[4, 5, 1, 2, 3]
[3, 4, 5, 1, 2]
```

Dry run →

First Rotation

arr = [7,9,10] , K=2
0 1 2 → index

[10, 9, 10] (Replace by 7)
0 1 2

[10, 7, 10]
0 1 2

[10,7,9]

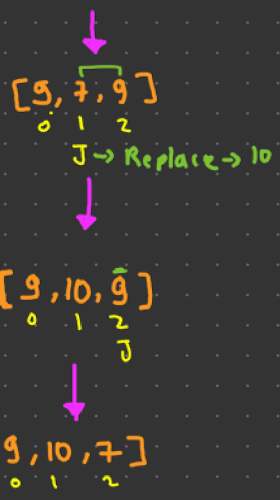
Here while loop Breake,

Result After 1st Rotation = [10,7,9]

In the same way 2nd Rotation happen.

2nd rotation

arr = [10, 7, 9],
0 1 2



Here while loop Breake
when J=3,

Finall Result after 2 time Rotation →

arr = [7, 9, 10]

① Rotation → arr = [10, 7, 9]

② Rotation ⇒ arr = [9, 10, 7]

Note: What if K > len(arr), then How Rotate the array ?

Ex → [1, 2, 3, 4, 5] where k = 50, Rotate the array.

9 Here [K > Size]

↳ k = 50 % 5 ⇒ 0

→ Here we no need to do anything